Space Carving MVD Sequences for Modeling Natural 3D Scenes

Youssef Alj, Guillaume Boisson, Philippe Bordes, Muriel Pressigout, Luce Morin

Space Carving MVD Sequences for Modeling Natural 3D Scenes

Youssef Alj[a,b], Guillaume Boisson[a], Philippe Bordes[a], Muriel Pressigout[b], Luce Morin[b]

[a]Technicolor, Cesson-Sévigné, France
[b]Institut d'Electronique et des Télécommunications de Rennes (IETR), Rennes, France

ABSTRACT

This paper presents a 3D modeling system designed for Multi-view Video plus Depth (MVD) sequences. The aim is to remove redundancy in both texture and depth information present in the MVD data. To this end, a volumetric framework is employed in order to merge the input depth maps. Hereby a variant of the Space Carving algorithm is proposed. Voxels are iteratively carved by ray-casting from each view, until the 3D model be geometrically consistent with every input depth map. A surface mesh is then extracted from this volumetric representation thanks to the Marching Cubes algorithm. Subsequently, to address the issue of texture modeling, a new algorithm for multi-texturing the resulting surface is presented. This algorithm selects from the set of input images the best texture candidate to map a given mesh triangle. The best texture is chosen according to a photoconsistency metric. Tests and results are provided using still images from usual MVD test-sequences.

Keywords: Multi-view Video plus Depth (MVD), 3DTV, FTV, depth map fusion, Space Carving, multi-view texture mapping.

1. INTRODUCTION

Multi-View Imaging (MVI) has gained an unceasingly growing interest in the last few years. Used with stereoscopic displays, MVI provides a realistic depth perception to the user and allows a virtual navigation around the scene. It also offers the possibility to synthesize virtual views, opening therefore a broad spectrum of research topics and applications such as 3DTV and Free-viewpoint TV (FTV). To this end, depth estimation from stereoscopic matching has been widely investigated. Nevertheless, several technical deficiencies hampered a successful introduction of these applications to the mass market. Actually an end-to-end MVI chain presents many challenges due to the inherent problems related to scene capture, data representation and transmission. Indeed, during scene acquisition, multiple cameras are used, hence several photometric errors may occur due to the changing illumination across different views, which increases the signal ambiguity during depth estimation. Scene representation is also a challenging task. One should strive to build a dense yet non-redundant scene model from the set of overlapping views. The compromise between model simplicity and easy rendering is usually hard to satisfy. The representation is then to be transmitted through a communication channel. Determining the proper rate-distortion trade-off is another key point. At decoder side, high fidelity to original views is required. Finally to assess the quality of virtual views, conventional image-based objective metrics such as Peak of Signal to Noise Ratio (PSNR) or Structural Similarity (SSIM) are not applicable since the reference images for such views are not available.

In the context of scene representation, research activities have been strengthened worldwide in order to overcome the issues discussed above. Scene representation methods can be classified into three categories : image-based representations, geometry-based representations and intermediate representations.

Image-based representations : Earlier attempts tried to extend image processing tools to 3D scene representation. A scene is simply represented using the set of captured images. This is the case in Multi View Video (MVV) representation. The 2D+Z[1] scheme uses a single view in addition to a depth image, also known as depth map, representing the depth information of each object of the scene. In this paper we consider Multi-view Video plus Depth representation (MVD)[2] as input, each view being composed of the captured image and the corresponding depth map.

Geometry-based representations : In the computer vision community, one of the core tasks is to extract 3D information from images. Thus many algorithms have been developed to extract a 3D model from multi-view data. On the other hand, the computer graphics community is developing IBR (Image Based Rendering) that is modeling of 3D scenes partly using real images as (one of the) input data. A 3D model extracted from the images may also be used for multi-view representation and compression. The extracted 3D model may be mesh-based,[3] voxel-based,[4] or point-based.[5]

Intermediate representations : Layered Depth Images[6] (LDI) were introduced in order to reduce redundancy in MVD material, by selecting a reference view and adding occluded information contained in adjacent views to the representation. 3D video billboards approach[7] extends the work developed in[8] and uses a hybrid approach combining depth maps and textured mapped rectangles intended to represent detailed geometry of the scene. Polygon soup representation was developed in[9] and uses a set of disconnecting and overlapping 3D polygons as a compact representation of the scene.

Scene representation has become an active research area with a wide variety of applications. Hence the choice of scene representation depends on the aimed application. For 3D video applications[10] , image-based representation are more popular since high fidelity with respect to original data is often required at the rendering stage. For mixed/augmented reality applications, geometric-based modeling methods are the most ubiquitous approaches. Typically, the purpose in such applications[11] is to recover an object of interest from a set of captured images. Foreground objects are extracted from input images, and it's not necessary to model the whole scene. High fidelity with respect to original views is therefore not necessarily satisfied, but they can achieve real time rendering with interactive user-experience.

In this paper, we take advantage of image-based representations, thanks to the use of MVD data, to ensure high fidelity to original data, and geometric-based representation to enforce the compactness of our scene representation.

The contributions of this paper are twofold. First, a new volumetric framework is proposed in order to merge the input depth maps into a single and compact surface mesh. This is the subject of section 3. Second, a new multi-view texture mapping using photoconsistency is introduced to texture the resulting surface mesh. This is presented in section 4. Last results are presented in section 5. But first in section 2 we will present a short state-of-the-art.

2. RELATED WORK

In this section, a short review of volumetric methods for scene representation is presented (see[12] and[13] for more detailed surveys).

Scene representation from a set of images has been an active research area for several decades. The aim is to build a 3D model given 2D views of the scene. Volumetric methods have known a successful introduction in computer vision community. Indeed they provide a robust and compact representation of the scene. Typically, these methods assume the existence of a bounding volume on which the scene of interest lies. This bounding volume is subdivided into cubical elements called *voxels*. During the process of *Space Carving* each voxel is labelled as *transparent* or *opaque*, depending on a consistency measure extracted from the input images. The volumetric representation of the scene is the set of opaque voxels. A surface mesh is then extracted from this volume using *Marching Cubes* algorithm. The consistency measure is based either on the silhouette information extracted from input images, which is referred to as *shape-from-silhouette* method, or according to the photometric information contained in the input images, which is referred to as *shape-from-photoconsistency* method. Alternative approaches use range images as input data and fuse them into a single surface.

Shape-from-silhouette : These methods require that foreground objects in the input images can be extracted from the background. Each object of interest is reconstructed from the set of silhouette images. Therefore the scene can be recovered by integrating each reconstructed objects with the background images. A silhouette image is a binary image where each pixel's value indicates whether the ray from the camera center to the input image

intersects an object of interest or not. The set of rays defines a silhouette cone. The intersection of the silhouette cones associated to all cameras is called the *visual hull*[14] . The visual hull of an object can also be defined as the maximal shape that gives the same silhouette as the actual object's silhouette for all views. A voxel is determined to be in the visual hull by projecting it into the set of all images and testing if it is inside the silhouette. To optimize scene space traversal, Potmesil[15] extends the regular-grid-based visual hull to octrees. Rather than using a voxel description, the authors in[11] use Delaunay tetrahedrization as a discretization of the scene space. The final surface is obtained by removing tetrahedrons lying outside the visual hull using the silhouette information. The major drawback of the shape-from-silhouette approach is that not all concavities present in the silhouette images can be represented.

Shape-from-photoconsistency : In such schemes, voxel consistency is measured directly using the captured images. This is the idea behind the Space Carving algorithm[16] . More precisely, a voxel is said to be photo-consistent with the set of input images if its projections onto the cameras in which this voxel is visible are all the same. In practice, a user-defined threshold is used in order to control the correlation between the projected voxels and the input data. This limitation has been overcome within the probabilistic framework developed by[17] . However, Space Carving suffers from another severe limitation. Indeed the surface to recover is assumed to be Lambertian. However natural scenes may contain areas with changing illumination effects due to image capturing conditions. Therefore, this assumption is usually not satisfied and this may significantly damage the relevance of the output volume by carving away wrong voxels. An approach combining the visual hull and photoconsistency constraints was developped by[18] . They formulated the problem of finding an optimal photoconsistent surface as an energy minimization problem. This cost function is discretized on a graph and the solution is obtained as the minimum cut solution of this graph.

Volumetric fusion of range images : Curless and Levoy proposed[19] a method for merging range images that relies on computing a weighted signed distance function cumulated respectively for each range image. The signed distance function takes 0-value for voxels containing points from acquired data. The distance function is computed from each voxel to the sensor and its value is stored for each voxel. For a given voxel, the so-called distance function is cumulated over each range image. Zero-crossing voxels correspond to the merged surface.

The proposed approach in this article is closer to range images fusion approaches since depth maps can be considered as range images. However as in the photo-consistency approach, the aim is to build a 3D model photoconsistent with the set of input images. The photoconsistency constraint is thus enforced using the proposed texture-mapping method.

3. GEOMETRY MODELING : A VOLUMETRIC FRAMEWORK FOR DEPTH MAPS FUSION

In this section, our volumetric framework intended to merge the input depth maps is presented. Considering a generic framework, the input sequences are calibrated but not necessarily registered. Epipolar lines may not be parallel to pictures lines. Consequently 3D reprojection operations may be more complex than horizontal shifts depending on objects' depth. The issue of the inter-view disparity estimation is not addressed in this paper, and the provided depth maps are assumed to be reliable. The purpose is to merge the input depth maps into a single surface mesh consistent with the original views.

An overview of our merging scheme is sketched in figure 1. First, for each provided depth map, a high resolution mesh is built. Second, the bounding volume enclosing the set of the built meshes is defined then discretized into voxels. A voxel may be either opaque or transparent. For each available camera, the set of voxels lying in its viewing cone are determined and labelled as opaque. The next step is dedicated to the Volumetric Mesh Hull (VMH) determination. Given a mesh corresponding to a viewpoint, the VMH defines the set of voxels that lie on this surface mesh. Our formulation of the Space Carving algorithm relies on the computed VMH. Indeed, for each available view, rays are cast from the camera center to each voxel of the corresponding VMH. The status of each voxel lying on this ray is then set to transparent and dedicated to be carved. Finally, a surface mesh is extracted from this binary-labelled volume using the Marching Cubes algorithm.

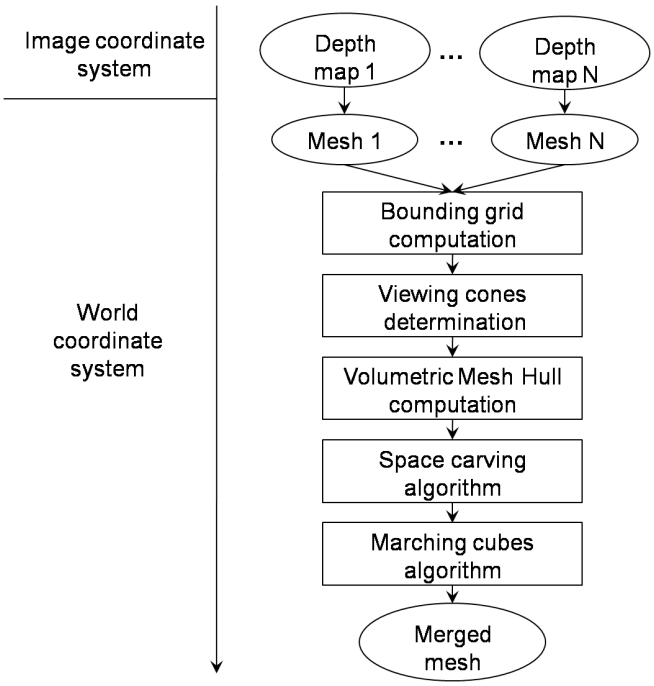

FIGURE 1. Our volumetric framework for depth maps fusion.

3.1 From depth maps to meshes

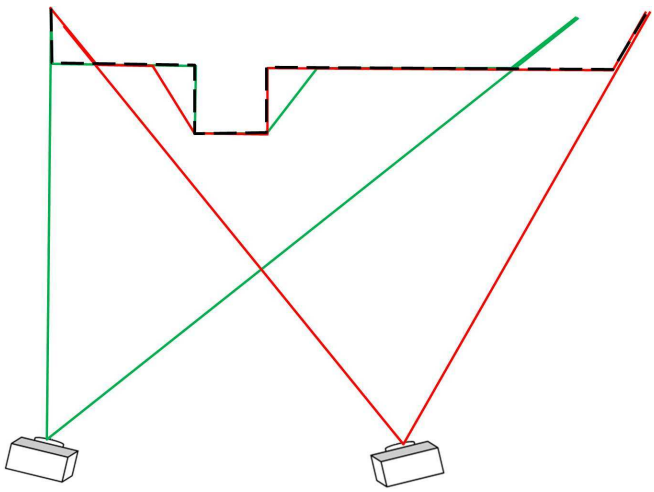

FIGURE 2. Example of two surfaces to be merged (one in green and the other in red). The expected surface as output of the algorithm is in black dotted line. Cameras viewing each mesh are also represented with their viewing cones.

The first step of our algorithm is the generation of one mesh for each of the provided depth maps. From each of the available views, a high resolution uniform mesh is built in the image domain. Each mesh vertex represents a pixel in the depth image. Registration is performed by converting each triangular mesh back to world coordinate system using equation (1).

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} & R & & T \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} f_x & S_{uv} & c_u & 0 \\ 0 & f_y & c_v & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} su \\ sv \\ s \\ 1 \end{pmatrix} \tag{1}$$

Where, X, Y and Z are the coordinates of a 3D point in the world reference Coordinate System. u and v are the coordinates of the projected 3D point in image coordinate system, and $s = z_{cam}$ is the depth of the pixel (u, v) with respect to the camera Coordinate System.
R and T respectively denote the rotation matrix and the translation vector of the considered camera's Coordinate System in the world Coordinate System. f_x, f_y, S_{uv}, c_u and c_v are the camera's parameters.

3.2 Bounding grid computation

The set of meshes are now available for our volumetric representation. The basic data structure on which our volumetric framework operates is a volumetric grid. This grid is built by a regular subdivision of the bounding box enclosing the set of meshes into parallelepipedic elements called voxels. A binary labelling strategy is used to label the voxels. Each voxel can be either opaque or transparent. The label of each voxel grid is initialized to transparent. Voxels labels are modified throughout the algorithm according to their relevance to represent the scene.

3.3 Viewing cones determination

In this step, the purpose is to find the set of voxels lying in each camera's viewing cone. Voxels outside this viewing cone are labelled as transparent. The viewing cone of each camera is modeled by a frustum defined by four planes bounding the range of visible points according to each axis (see Figure 3). The task is then to compute for each camera the set of voxels that lie in its corresponding frustum. To this end, frustum planes equations are determined for each camera, these equations allow to perform the inside/outside test for each voxel center against the considered frustum.

Frustum planes equations : Equation (1) represents the conversion from image to world coordinates system. This conversion is invertible, and image coordinates can be expressed using the following equation :

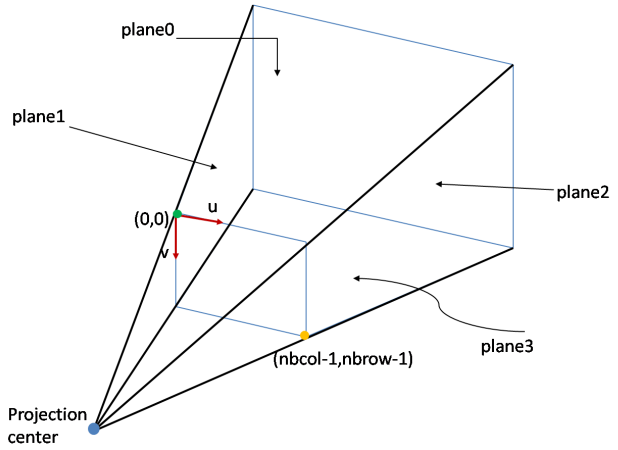

FIGURE 3. Frustum planes computation.

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{pmatrix} f_x & S_{uv} & c_u & 0 \\ 0 & f_y & c_v & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} & R^{-1} & & -R^{-1}.T \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix},$$

This equation can be reformulated by computing the product of the two central matrices :

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix},$$

The last equation leads to the two following parametric plane equations :

$$X(p_{11} - up_{31}) + Y(p_{12} - up_{32}) + Z(p_{13} - up_{33}) + p_{14} - up_{34} = 0 \quad (2)$$

$$X(p_{21} - vp_{31}) + Y(p_{22} - vp_{32}) + Z(p_{23} - vp_{33}) + p_{24} - vp_{34} = 0 \quad (3)$$

Equation (2) with $u = 0$ (resp. $u = \text{nbcol} - 1$) is the equation $E_1(X, Y, Z)$ of the vertical left-hand plane 1 (resp. the equation $E_2(X, Y, Z)$ of the right-hand plane 2) see figure 3.

Equation (3) with $v = 0$ (resp. $v = \text{nbrow} - 1$) determines the equation $E_3(X, Y, Z)$ of the upper plane 0 (resp. the equation $E_4(X, Y, Z)$ of the bottom plane 3).

Once planes equations are computed, inside/outside points with respect to each camera can be determined as the following : Given a point in the frustum $I(X_I, Y_I, Z_I)$

Inside condition test : $\forall i,\ 1 \leq i \leq 4$ compute $\text{sign}(E_i(X_I, Y_I, Z_I))$ where,

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Given a voxel defined by its center $V(X_V, Y_V, Z_V)$, if $\forall i,\ 1 \leq i \leq 4$,
$\text{sign}(E_i(X_I, Y_I, Z_I)) = \text{sign}(E_i(X_V, Y_V, Z_V))$ then this voxel is inside the frustum, otherwise this voxel is outside.

At the end of this step, all voxels belonging to the union of cameras viewing cones are labelled as opaque. All the remaining voxels within the bounding grid are still labelled transparent.

3.4 Building the Volumetric Mesh Hull (VMH)

This section describes the computation of the Volumetric Mesh Hull (VMH) which will be used during the space carving process. For each input mesh, the set of voxels intersecting the surface mesh defines the VMH associated with this mesh.

As an output each voxel has an additional information defining which VMH(s) this voxel belongs to.

For each mesh, each triangle is visited and scanned with respect to the voxel grid, i.e. this scan defines the voxels lying on this triangle. All these voxels are then marked to belong to the current VMH by updating the additional information associated with the voxel. In order to scan the 3D triangle surface, the classical 2D scan line algorithm is extended to the 3D case.

The 3D scan line algorithm is presented in algorithm 1.

```
//  Triangles traversal
for each triangle T do
    for each axis X, Y and Z do
        Compute the integer bounds b_min and b_max of T along the axis considered.
        for m ← b_min to b_max do
            Compute the intersection line of triangle T with the plane at coordinate m.
            Find voxels along this intersection line using 3D Bresenham line algorithm.
            Add these voxels to VMH.
        end
    end
end
```

Algorithm 1: Volumetric Mesh Hull (VMH) determination.

3.5 Space carving

Until now, an opaque voxel is a voxel lying in at least one camera's viewing cone. A VMH voxel is a voxel lying on one of the surface mesh. In this step, space carving is performed by updating voxels labels from opaque to transparent for all voxels "in front of each VMH". More precisely, each VMH is considered and a geometric consistency criterion is employed in order to update voxels labels. This criterion is based on the relative position of the voxel according to the considered surface mesh and the corresponding viewpoint. A voxel is said to be geometrically consistent with a surface mesh if it lies behind it with respect to the viewing camera. Voxels detected as geometrically inconsistent are labelled as transparent. The carving process is performed by iteratively updating the carved volume with respect to each camera. The volume to be carved is initialized to the union of voxels inside the set of viewing cones computed in section 3.3 (see figure 4). This volume is then refined iteratively with geometrical information lying in each view (see figures 4 and 5), carving occluding areas relative to each camera. For each of the available views, rays are cast from the viewpoint to each voxel of the camera's corresponding VMH. For each ray, voxels lying on the current ray are scanned using 3D Bresenham line retracing algorithm. Since they are geometrically inconsistent, these voxels are dedicated to be carved (i.e. updated to transparent).

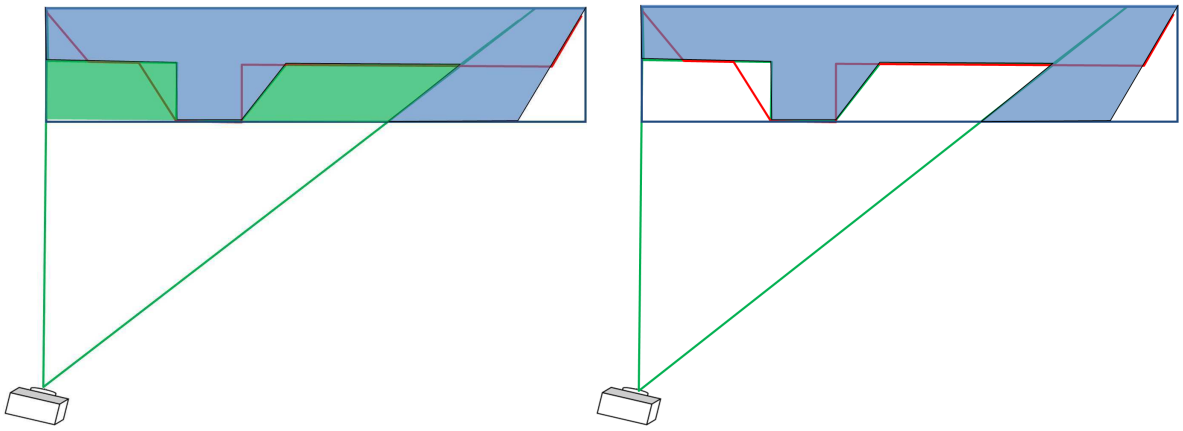

FIGURE 4. Space carving according to the left camera. Rays are cast from the camera center to the corresponding VMH (left). Green areas are carved. The result of this carving operation according to the green camera is shown on the right.

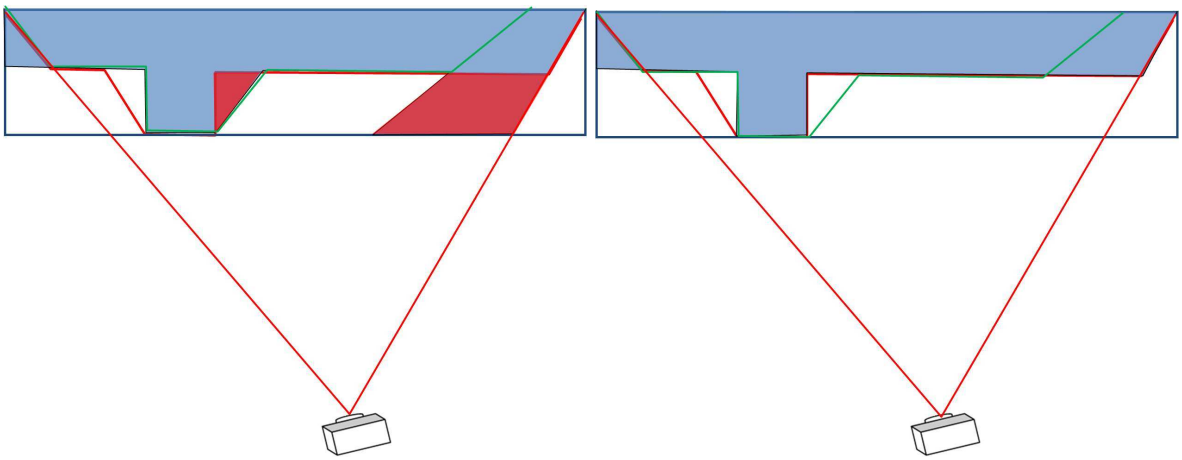

FIGURE 5. Space carving according to the right camera. For the carved volume to be updated, rays are cast from the right camera to its corresponding VMH (left). Red areas are carved. The result of this carving step is shown in the right image.

```
// Views traversal
for each camera do
    Get the camera position.
    Get the corresponding VMH.
    for each voxel in the VMH do
        Find the 3D Bresenham line between camera's position and voxel's center.
        Update each of these voxels' labels to transparent.
    end
end
```

Algorithm 2: Space carving algorithm.

At the end of the space carving step, the set of opaque voxels defines the volumetric model for our MVD data.

3.6 Marching Cubes

Finally, the merged surface mesh is then extracted from our binary-labelled volumetric representation using the Marching Cubes algorithm. Marching Cubes algorithm was originally introduced in[20] . It takes as input a regular volumetric data set. Such a data set has a scalar value assigned to each voxel vertex. During processing, each cube vertex that has a value less or equal than a predefined isovalue, is marked. All other vertices are left unmarked. Consequently the Marching Cubes algorithm outputs the triangular mesh connecting the marked vertices.

4. MULTI-VIEW TEXTURE MAPPING

The photoconsistency constraint is enforced by texturing the merged surface mesh extracted from Marching Cubes algorithm. Input texture images of the MVD data are used in order to texture the merged mesh. To this end, a new texture mapping algorithm is proposed based on a photoconsistency metric. The main steps of this algorithm are summarized in figure 6. First, visibility is determined for each triangle in the merged mesh. Second, each available texture is mapped on visible triangles of this merged mesh. Photometric projection error is computed for each triangle and for each input view and stored in error images. The best texture for each triangle is chosen as the one that minimizes a photoconsistency metric. Triangles not visible by any camera are textured with intermediate view. Synthesized views are produced by rendering the textured mesh, each triangle being textured with the best texture according to the photo consistency metric.

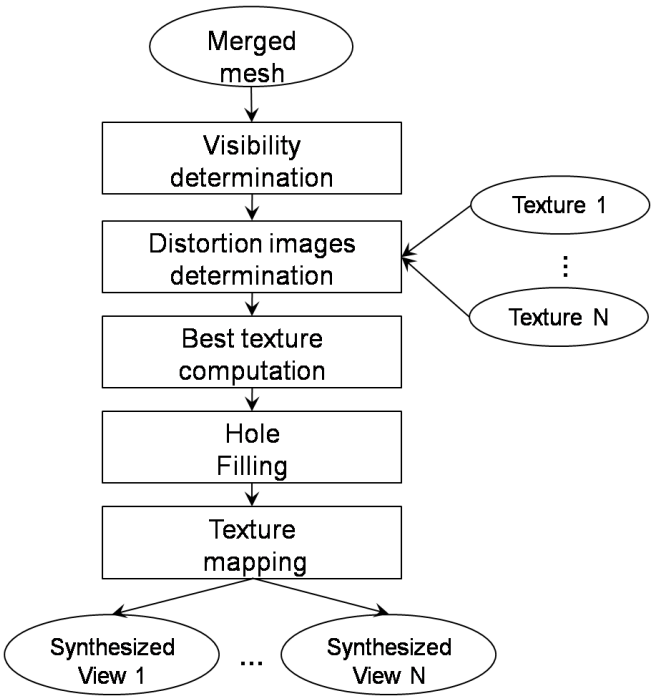

FIGURE 6. The overall framework of our texture mapping based on photoconsistency metric.

Given a set of texture images $\mathcal{I} = \{I_1, \ldots, I_n\}$ and a triangular mesh $\mathcal{M}$, the aim is to find for each mesh triangle $T \in \mathcal{M}$ the best texture image $\hat{I}_T \in \mathcal{I}$. Texturing the triangle T is formulated as an energy minimization problem. The best texture for T is found by minimizing a cost function formulated in terms of how much the triangle T is photoconsistent with the set of input images in which the triangle T is visible. Photo-consistency is estimated by projecting the textured mesh onto the views $\mathcal{V} = \{V_1, \ldots, V_n\}$ corresponding to input images $\mathcal{I}$.

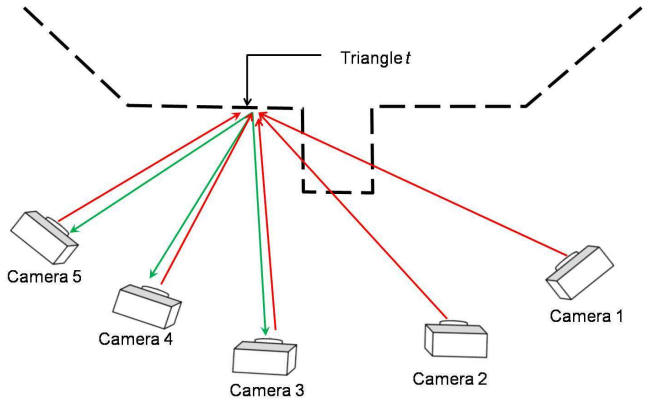

FIGURE 7. Texture mapping using the photoconsistency metric. For sake of simplicity, the surface mesh is shown with the black dotted line representing the mesh triangles. The triangle t is visible in cameras 3, 4, 5. For each texture $I_i \in \mathcal{I}$, the triangle t is textured with texture I_i (red arrow for texture mapping operation) and projected (green arrow for projection operation) onto the cameras 3, 4, and 5 - i.e. the set of cameras in which t is visible. The quadratic errors between the textured triangle and each of its projection on images I_3, I_4 and I_5 are computed. The error (i.e. the photoconsistency metric) of texturing the triangle t with the texture I_i is the sum of all these quadratic errors. The best texture is the one that minimizes the photoconsistency metric.

4.1 Visibility determination

From the merged mesh, visibility for each triangle is determined in order to perform texture mapping. The status of each mesh triangle is initialized to visible. Triangle visibility with respect to each camera is determined using OpenGL z-buffer. In the first pass, the input mesh is projected onto the current view, and the z-buffer is extracted. The second pass is dedicated to visibility determination using the computed z-buffer. Each mesh vertex is projected onto the current view and the depth component, denoted as $z_{\text{projected}}$, is checked against the pixel depth z_{buffer}. If the projected vertex is behind the pixel in the z-buffer, then this vertex is hidden, and thus the set of mesh triangles lying on this vertex are marked hidden. The pseudo-code is provided in algorithm 3.

```
//  Views traversal
for each view V_j do
    Initialize all triangles to visible.
    Project M onto V_j.
    Read z_buffer.
    // Mesh vertices traversal
    for each vertex v do
        Determine (q, l, z_projected) the projection of the vertex v(X, Y, Z) onto V_j.
        if z_buffer[q, l] > z_projected then
            v is a hidden vertex. Mark all the triangles lying on v as hidden.
        end
    end
end
```

Algorithm 3: Visibility determination

4.2 Distortion image computation

In order to determine the error of texturing a triangle with an input texture I_i, each available texture in then mapped on the merged mesh and projected onto each camera. Let $\mathcal{M}_{I_i \to V_j}$ be the projection onto the view V_j of the mesh $\mathcal{M}$ textured with the image I_i. For each available texture the following distortion image is computed :

$$D_{i,j} = \|\mathcal{M}_{I_i \to V_j} - I_j\|_2$$

```
// Textures traversal
for each texture I_i do
    // Views traversal
    for each view V_j do
        Compute M_{I_i → V_j} : projection of the mesh textured with texture I_i onto view V_j.
        D_ij = ‖ M_{I_i → V_j} − I_j ‖_2
    end
end
```

Algorithm 4: Compute distortion images.

4.3 Best texture determination

Among all the available textures the best texture image is chosen for each mesh triangle, using the distortion images. Let M_{T,V_j} be the projection of the triangle T onto the view V_j.

The photoconsistency metric measures the error of texturing the triangle T with the texture image I_i. $\forall i \in \{1, \ldots, n\}$ we compute :

$$\text{Error}_{T,I_i} = \sum_{\substack{j=1 \\ T \text{ visible in } V_j}}^{n} \left(\sum_{(q,l) \in M_{T,V_j}} D_{i,j}[q,l] \right) , \tag{4}$$

The best texture for a triangle T is then given by :

$$\hat{I}_T = \arg\min_{I_i \in \mathcal{I}} \text{Error}_{T,I_i} . \tag{5}$$

```
// Triangles traversal
for each triangle T do
    // Views traversal
    for each view V_j do
        Determine M_{T,V_j} the projection of the triangle T onto V_j where T is visible.
    end
    // Textures traversal
    for each texture I_i do
        Compute Error_{T,I_i}.
    end
    Determine Î_T.
end
```

Algorithm 5: Compute the best texture for each triangle.

4.4 Texture mapping

Finally, pictures are rendered. The 3D model is projected according to the virtual camera parameters supplied by the user. Then each triangle is textured using the texture coordinates determined during the above-mentioned texture mapping step.

4.5 Hole filling

Actually there are some triangles remaining whose all vertices are not visible simultaneously in the same view. These triangles can therefore obviously not be textured the usual way. In the current status of the modeling chain, these triangles are textured with texture coordinates from the central view, though it is not satisfying. Perspectives regarding this issue are twofold. First the final mesh could be warped for the number of such triangles be reduced. Second the remaining ones shall be inpainted in a 2D sense.

5. RESULTS

Results are presented for two sequences. The sequence *Breakdancers*, provided by Microsoft and shared in MPEG's former FTV group, presents 8 cameras arranged in the shape of an arc, with high-quality depth maps. The sequence *Balloons* was shot with a parallel setup. Rectified views and corresponding disparity maps were provided by Nagoya University and shared in MPEG's current standardization activities for 3DV technologies as a three-view test-sequence. The quality of *Balloons* depth maps is a bit lower than those of *Breakdancers*, but might be more realistic for practical applications. Both sequences present XGA resolution (1024x768).

5.1 Geometric modeling

The following figure illustrates the output surface of our geometric modeling scheme with a medium voxel resolution.

Geometric artifacts around sharp depth transitions are noticeable. At such resolution depth quantization artifacts also occur on plane surfaces. Both are expected to be lessened thanks to our texture mapping algorithm.

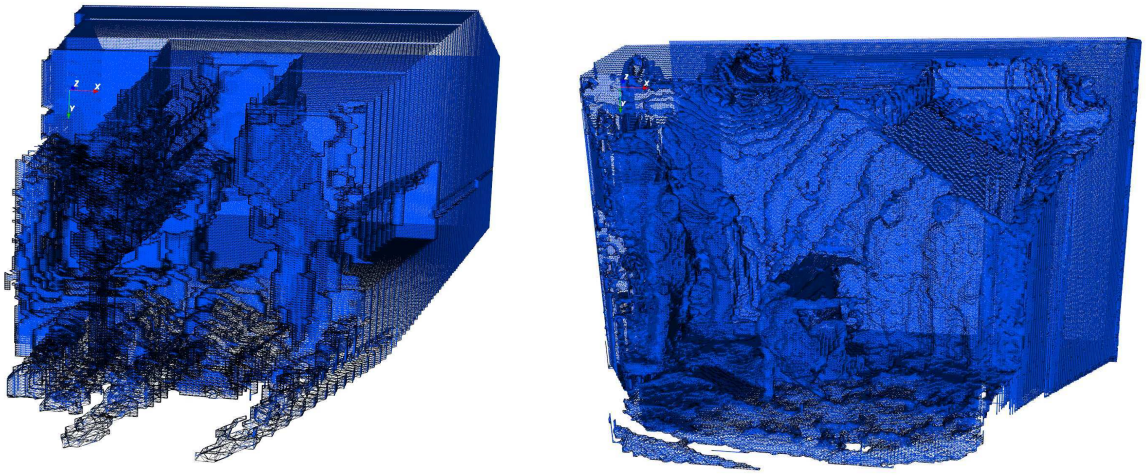

FIGURE 8. Results of our volumetric merging scheme using a volumetric resolution 250x250x250. Left : *Balloons* 3D mesh model. Right, *Breakdancers* 3D mesh model.

5.2 Texture mapping

Distortion computation : Distortion is computed between the rendered image and the original views using the PSNR metric. Note that distortion is computed only on triangles visible from at least one camera.

Camera	1	3	5
PSNR (dB)	28.9477	31.8392	30.9251

TABLE 1. Distortion of synthesized images for *balloons* sequences.

Camera	0	1	2	3	4	5	6	7
PSNR (dB)	28.6302	29.7552	31.2625	31.2188	31.7577	30.8564	29.863	29.2547

TABLE 2. Distortion of synthesized images for *breakdancers* sequences.

Though the distortion in terms of quadratic error is quite large, results are rather satisfying in terms of visual quality. Thanks to our texture mapping algorithm, our 3D modeling scheme is rather robust to input depth maps quality. Yet it is obvious that accurate registration and estimation helps in restoring ground truth in the geometric step.

6. CONCLUSION

In this paper we addressed the issue of 3D modeling natural video sequences from Multi-view Video plus Depth (MVD) material, without any assumption on the camera setup arrangement. A new geometrical modeling scheme was presented that merges the input depth maps into a single triangular mesh. To this end we designed a volumetric framework relying on an iterative Space Carving algorithm. Initial data are turned into voxels whose status (empty or non-empty) are updated according to their geometric consistency with the input depth maps. The frontier of the final volumetric model is turned back in a 3D mesh surface with Marching Cubes algorithm. A multi-view texturing scheme is also presented. This triangle-based algorithm assigns optimal textures coordinates to vertices of the final mesh. This is achieved by minimising color distortions between rendered pictures and the original views available at modeler side. Our 3D model consisting of both geometry and texture enables to synthesize any view around the initial cameras with few visual artifacts. As a perspective we plan to investigate the transmission issue, by reducing the coding cost of the geometrical model and formalizing the final texture signal to be delivered along.

FIGURE 9. Results of the proposed photoconsistency-based texture mapping on balloons sequences. Top left, reference texture image corresponding to camera 1. Top right, reference texture image corresponding to camera 5. Bottom left, synthesized image corresponding to camera 1. Bottom right, synthesized image corresponding to camera 5.

REFERENCES

[1] Fehn, C., Kauff, P., De Beeck, M., Ernst, F., Ijsselsteijn, W., Pollefeys, M., Van Gool, L., Ofek, E., and Sexton, I., "An evolutionary and optimised approach on 3d-tv," in [*Proc. of IBC*], **2**, 357–365 (2002).

[2] Merkle, P., Smolic, A., Muller, K., and Wiegand, T., "Multi-view video plus depth representation and coding," in [*Image Processing, 2007. ICIP 2007. IEEE International Conference on*], **1**, I–201, IEEE (2007).

[3] Balter, R., Gioia, P., and Morin, L., "Scalable and efficient video coding using 3-d modeling," *Multimedia, IEEE Transactions on* **8**(6), 1147–1155 (2006).

[4] Mueller, K., Smolic, A., Merkle, P., Kaspar, B., Eisert, P., and Wiegand, T., "3d reconstruction of natural scenes with view-adaptive multi-texturing," in [*3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*], 116–123, IEEE (2004).

[5] Waschbüsch, M., Würmlin, S., Cotting, D., Sadlo, F., and Gross, M., "Scalable 3d video of dynamic scenes," *The Visual Computer* **21**(8), 629–638 (2005).

[6] He, L., Shade, J., Gortler, S., and Szeliski, R., "Layered depth images," in [*Proceedings of the 25th annual conference on computer graphics and interactive techniques (SIGGRAPH 1998), July*], 19–24 (1998).

[7] Waschbüsch, M., Würmlin, S., and Gross, M., "3d video billboard clouds," in [*Computer Graphics Forum*], **26**(3), 561–569, Wiley Online Library (2007).

[8] Décoret, X., Durand, F., Sillion, F., and Dorsey, J., "Billboard clouds for extreme model simplification," in [*ACM Transactions on Graphics (TOG)*], **22**(3), 689–696, ACM (2003).

[9] Colleu, T., Pateux, S., Morin, L., and Labit, C., "A polygon soup representation for multiview coding," *Journal of Visual Communication and Image Representation* **21**(5-6), 561–576 (2010).

FIGURE 10. Results of the proposed photoconsistency-based texture mapping on Microsoft *breakdancers* sequences. Top left, reference texture image corresponding to camera 0. Top right, reference texture image corresponding to camera 7. Bottom left, synthesized image corresponding to camera 0. Bottom right, synthesized image corresponding to camera 7.

[10] McMillan, L. and Bishop, G., "Plenoptic modeling : An image-based rendering system," in [*Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*], 39–46, ACM (1995).

[11] Franco, J. and Boyer, E., "Efficient polyhedral modeling from silhouettes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **31**(3), 414–427 (2009).

[12] Dyer, C., "Volumetric scene reconstruction from multiple views," *Foundations of Image Understanding* , 469–489 (2001).

[13] Slabaugh, G., Culbertson, B., Malzbender, T., and Schafer, R., "A survey of methods for volumetric scene reconstruction from photographs," in [*International Workshop on Volume Graphics*], **2**(7), Citeseer (2001).

[14] Martin, W. and Aggarwal, J., "Volumetric descriptions of objects from multiple views," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (2), 150–158 (1983).

[15] Potmesil, M., "Generating octree models of 3d objects from their silhouettes in a sequence of images," *Computer Vision, Graphics, and Image Processing* **40**(1), 1–29 (1987).

[16] Kutulakos, K. and Seitz, S., "A theory of shape by space carving," *International Journal of Computer Vision* **38**(3), 199–218 (2000).

[17] Broadhurst, A., Drummond, T., and Cipolla, R., "A probabilistic framework for space carving," in [*Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*], **1**, 388–393, IEEE (2001).

[18] Vogiatzis, G., Torr, P., and Cipolla, R., "Multi-view stereo via volumetric graph-cuts," in [*Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*], **2**, 391–398, IEEE (2005).

[19] Curless, B. and Levoy, M., "A volumetric method for building complex models from range images," in [*Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*], 303–312, ACM (1996).
[20] Lorensen, W. and Cline, H., "Marching cubes : A high resolution 3d surface construction algorithm," *ACM Siggraph Computer Graphics* **21**(4), 163–169 (1987).